CUG 2004

Matrix Solution Software Out-of-Core Complex Valued Lower-Upper Decomposition

Marianne Spurrier *and* **Joe Swartz**, *Lockheed Martin Corp. and* **Bruce Black**, *Cray Inc*.

LOCKHEED MARTIN

CRAY

CUG 2004

Introduction

$$Ax = b$$

$$Ax = LUx = L(Ux) = Ly = b$$

$$Ux = y$$

- The decomposition poses an interesting I/O problem when the matrix becomes too large to fit in memory
- We investigate two different disk storage formats: Slabs and Blocks
- Development occurred on AHPCRC's X1 and our X1

CUG 2004

Slab LU Algorithm

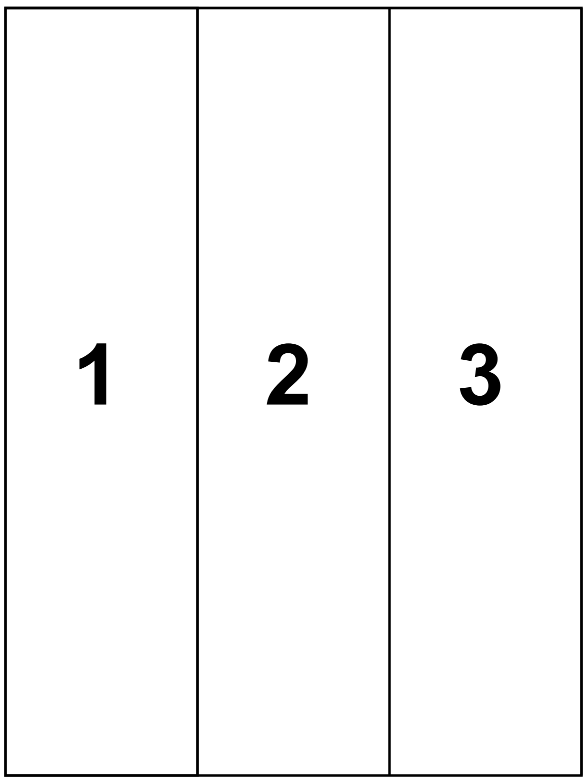
- The Slab LUD and Solve are based on work done by benchmark group in Cray Research, Inc.
  - first completed around 1988 to support a government customer
- Subsequent versions were maintained by CRI until 1998 when LMC Program, CSCF, began maintaining code for internal use

LOCKHEED MARTIN

CRAY

Basic Slab LU Algorithm

- Uses buffered asynchronous I/O (AIO) and BLAS routines



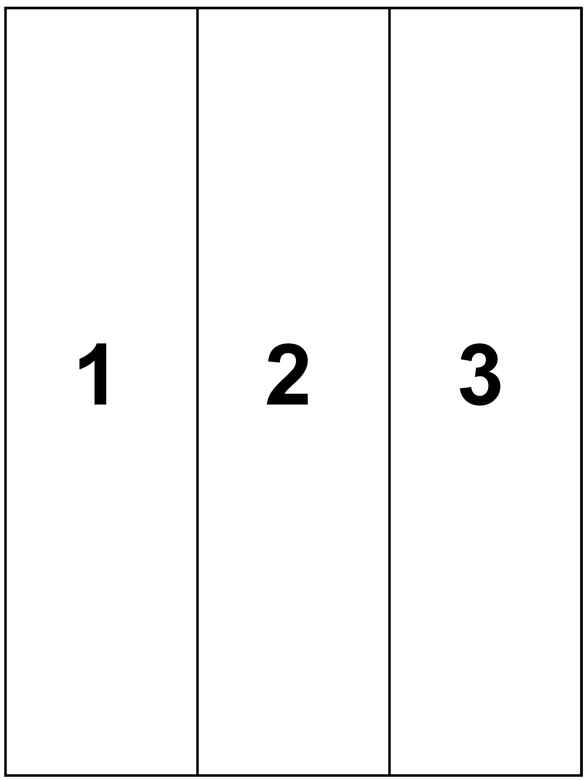
- One buffer is used to hold slab M to be worked on
- The other two are used to hold slabs 1 through M-1 needed to decompose slab M
- Alternate between computations and I/O

Basic Slab LU Algorithm

- For slab M of the matrix:
  - Read slab M into a buffer
  - Read slab 1 into one of the other buffers
  - Schedule the read of slab 2 into the third buffer
  - Compute using slabs 1 and M
  - Schedule the read of slab 3 into the buffer space that slab 1 held
  - Compute using slabs 2 and M
  - Repeat until done
  - Compute on slab M itself
  - Write out the result and start reading slab M+1

Basic Slab Solve Algorithm

- Uses buffered Asynchronous I/O (AIO) and BLAS routines



- One buffer is used to hold a slab of right hand side vectors
- The other two are used to hold the matrix slabs
- Alternate matrix slab buffers between computations and I/O both forward and backward solve

Basic Slab Solve Algorithm

- For a upper solution of RHS slab:
  - Read RHS slab into a buffer
  - Read slab 1 into one of the other buffers
  - Schedule the read of slab 2 into the third buffer
  - Compute partial upper solution with slab 1
  - Schedule the read of slab 3 into the buffer space that slab 1 held
  - Compute partial upper solution with slab 2
  - Repeat for all matrix slabs
  - Write out the result and start reading in the next RHS slab

Basic Slab Solve Algorithm

- For a lower solution of RHS slab:
  - Read RHS slab into a buffer
  - Read slab N into one of the other buffers
  - Schedule the read of slab N-1 into the third buffer
  - Compute partial lower solution with slab N
  - Schedule the read of slab N-2 into the buffer space that slab N held
  - Compute partial upper solution with slab N-1
  - Repeat for all matrix slabs
  - Write out the result and start reading in the next RHS slab

CUG 2004

LUD/Solve Algorithm Times

- For contested runs on a single MSP
  - 133,000 complex unknowns
    - 4 problems running concurrently on single MSPs
    - 2.7 gigabytes of buffer memory
    - 208 hours wall clock
  - 77,500 complex unknowns
    - 8 problems running concurrently on single MSPs
    - 2.7 gigabytes of buffer memory
    - 35 hours wall clock

LOCKHEED MARTIN
CRAY

Slab LU Algorithm Comments

- Slab algorithm minimized disk writes to one per slab
- Slab algorithm provides almost optimal I/O for the LUD computation
- Slab algorithm requires double the I/O actually needed to solve against the RHS vectors

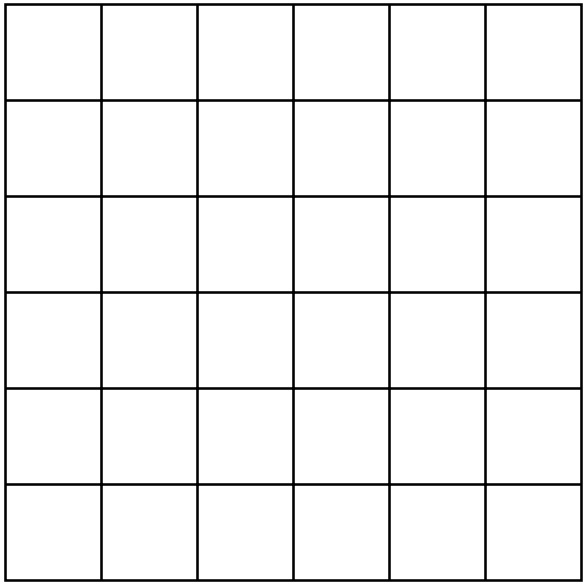
Slab Work in Progress

- Multi-MSP LU and Solve approaches minimize disk I/O by passing slabs between MSPs
  - Two approaches for passing slabs between MSPs in evaluation
- Algorithms in work

CUG 2004

Block LU Algorithm

- Uses buffered Asynchronous I/O (AIO) and BLAS routines (CGEMM, CTRSM, CSCAL, CTRSV, CGEMV)

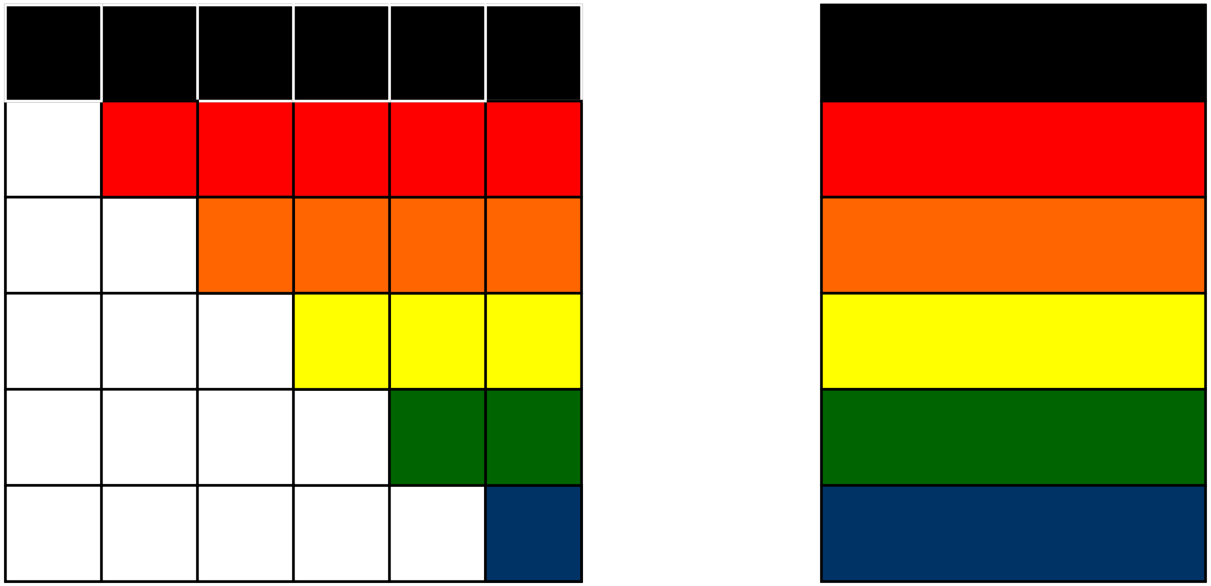


LOCKHEED MARTIN

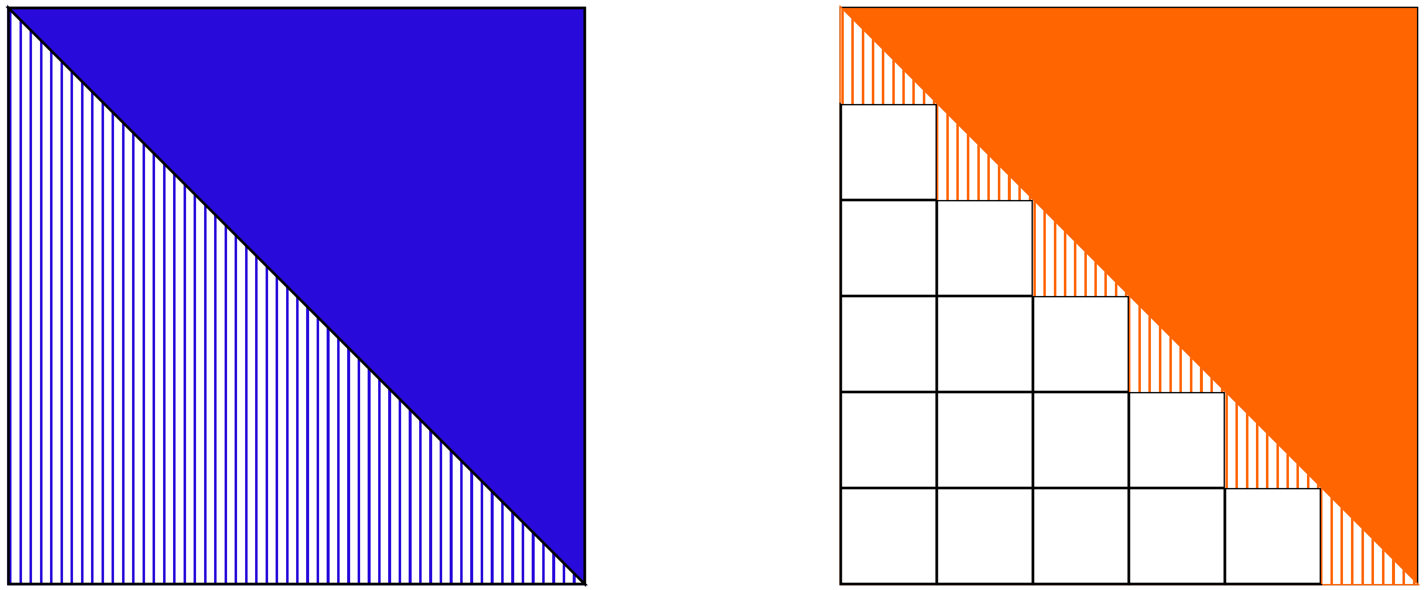
CRAY

Block Solve Algorithm

- Uses buffered (AIO) and BLAS routines (CGEMM, CTRSM)

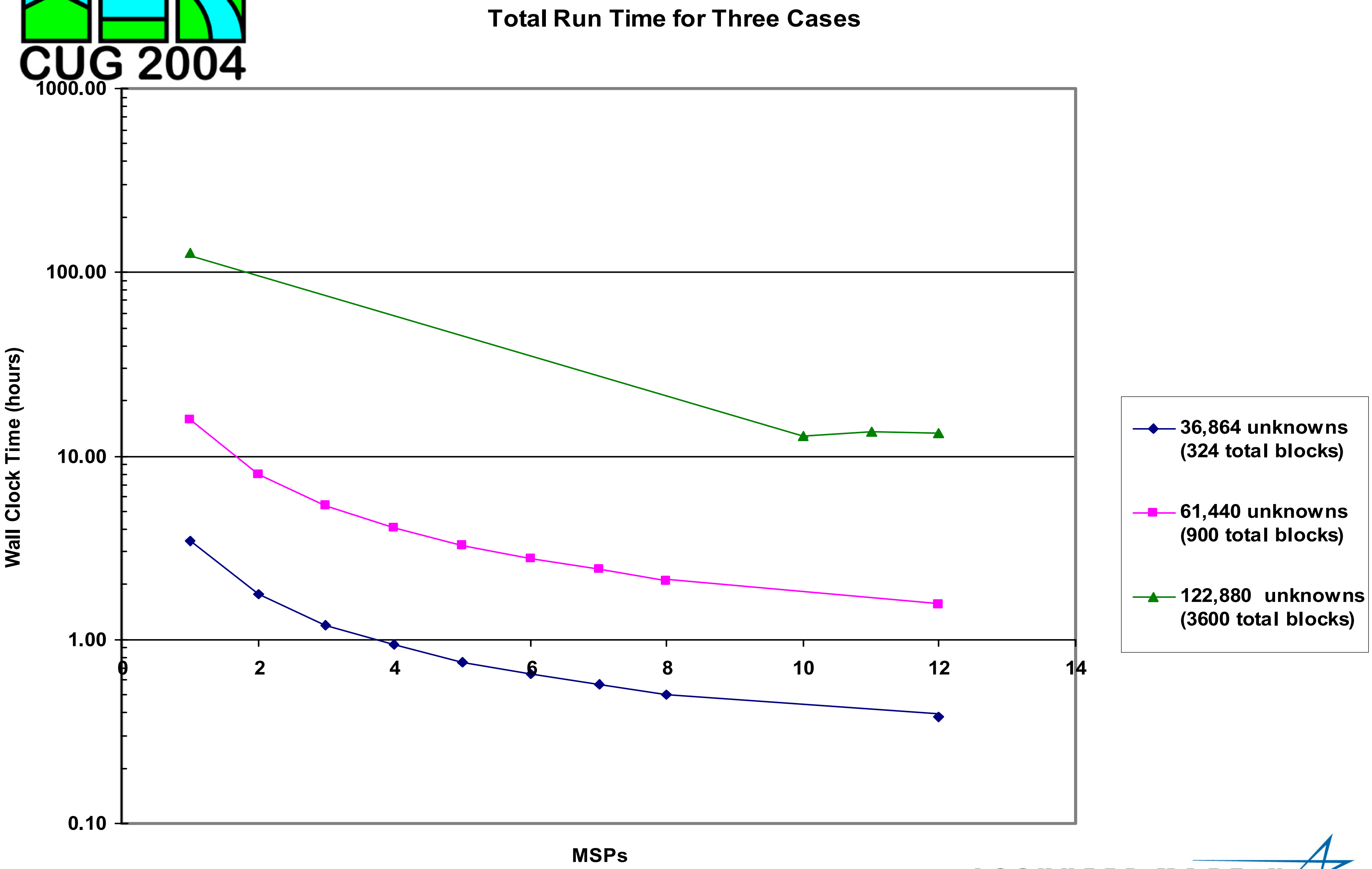


Comparison of I/O for Block and Slab Solvers



Block LU Benchmarks

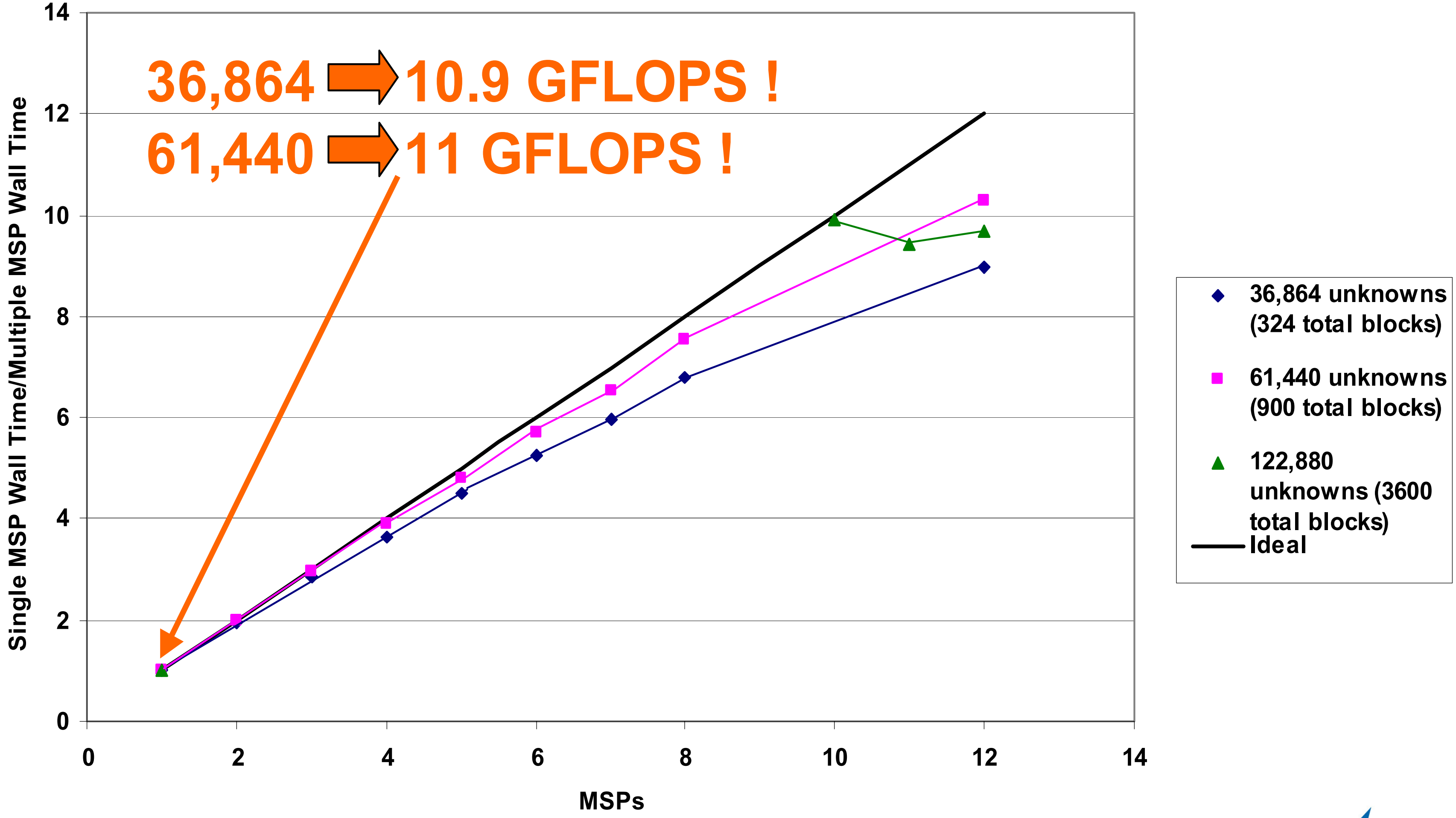
CUG 2004



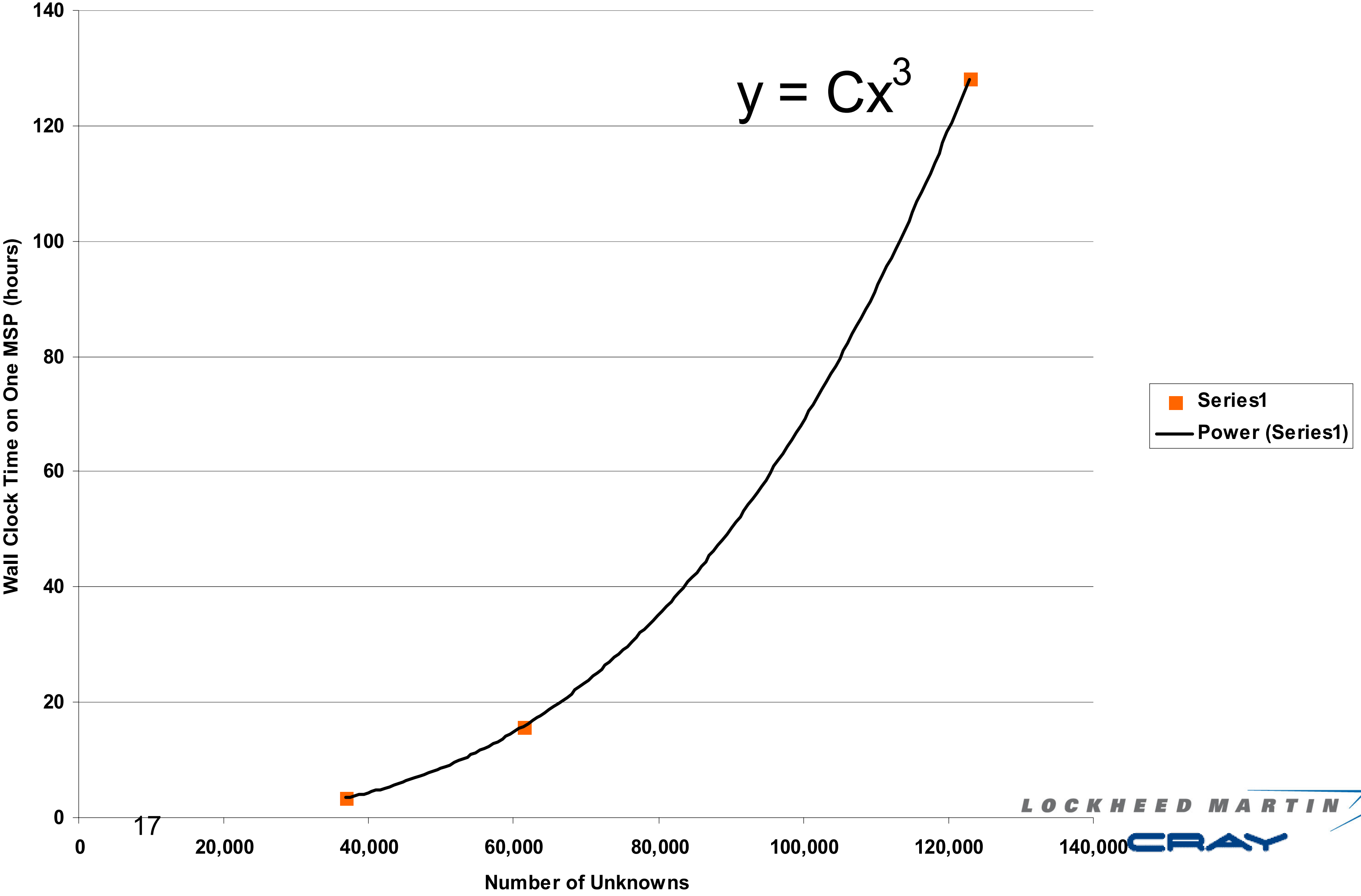
LOCKHEED MARTIN

CRAY

Scaling


36,864 ➡ 10.9 GFLOPS !
61,440 ➡ 11 GFLOPS !
Single MSP Wall Time/Multiple MSP Wall Time
MSPs
0
2
4
6
8
10
12
14
36,864 unknowns (324 total blocks)
61,440 unknowns (900 total blocks)
122,880 unknowns (3600 total blocks)
Ideal

N^3 Scaling

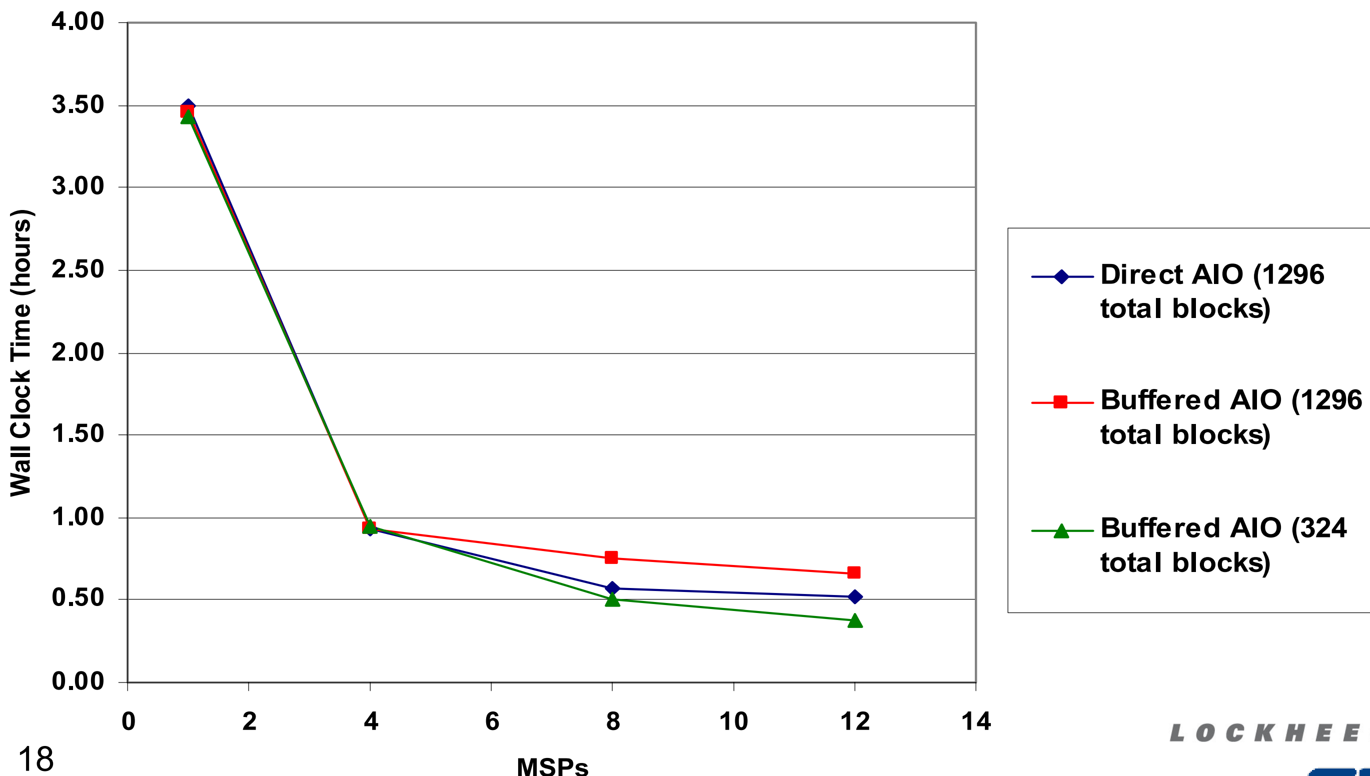


$y = Cx^3$
Wall Clock Time on One MSP (hours)
Number of Unknowns
0
20
40
60
80
100
120
140
0
20,000
40,000
60,000
80,000
100,000
120,000
140,000
Series1
Power (Series1)

Comparison of C I/O, AIO and Direct AIO for 36,864 Unknowns

CUG 2004

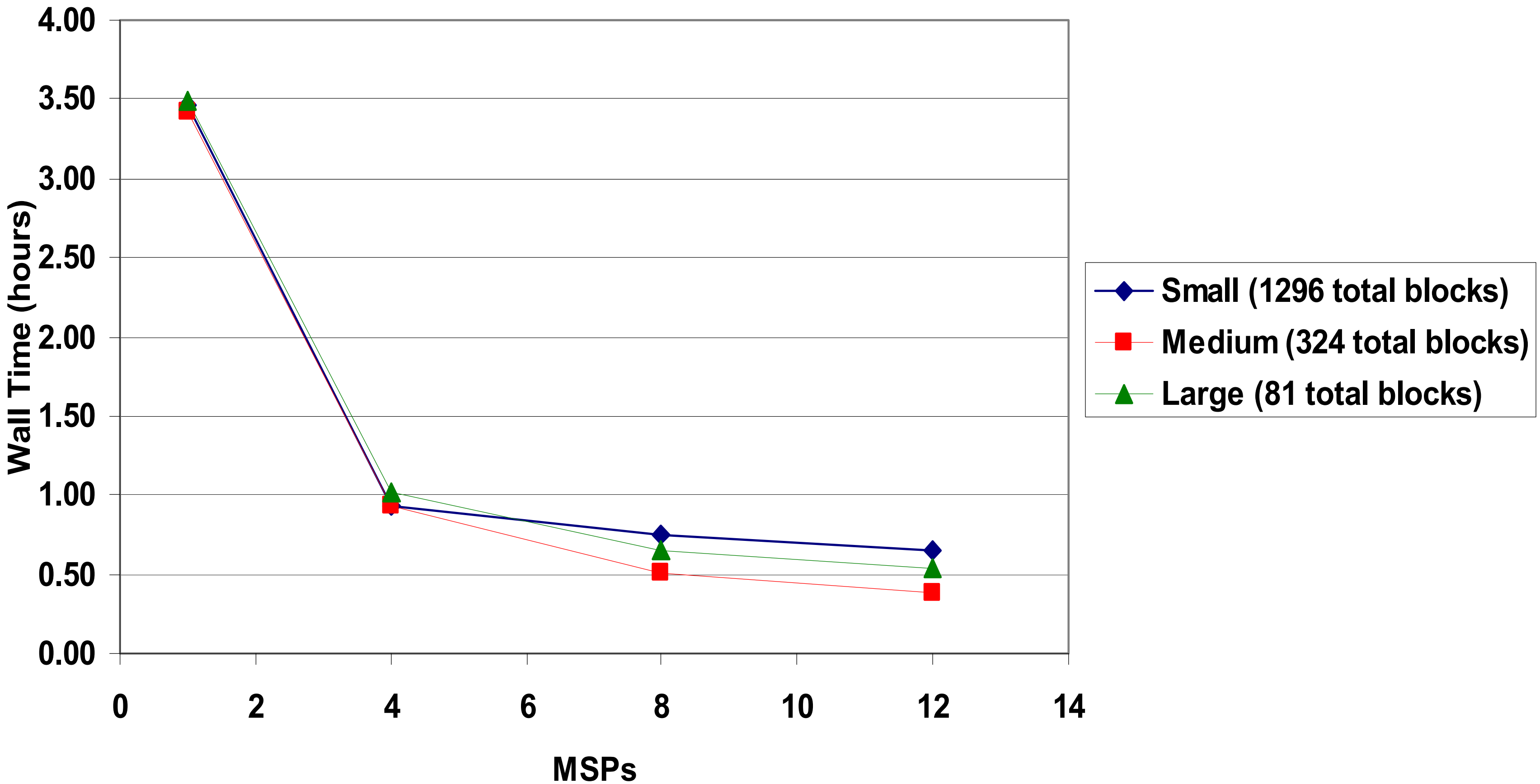
When AIO is used to overlap I/O and computations, it is approximately 22% faster than using synchronous C I/O



LOCKHEED MARTIN

CRAY

Changing the Block Size – 36,864 Unknowns

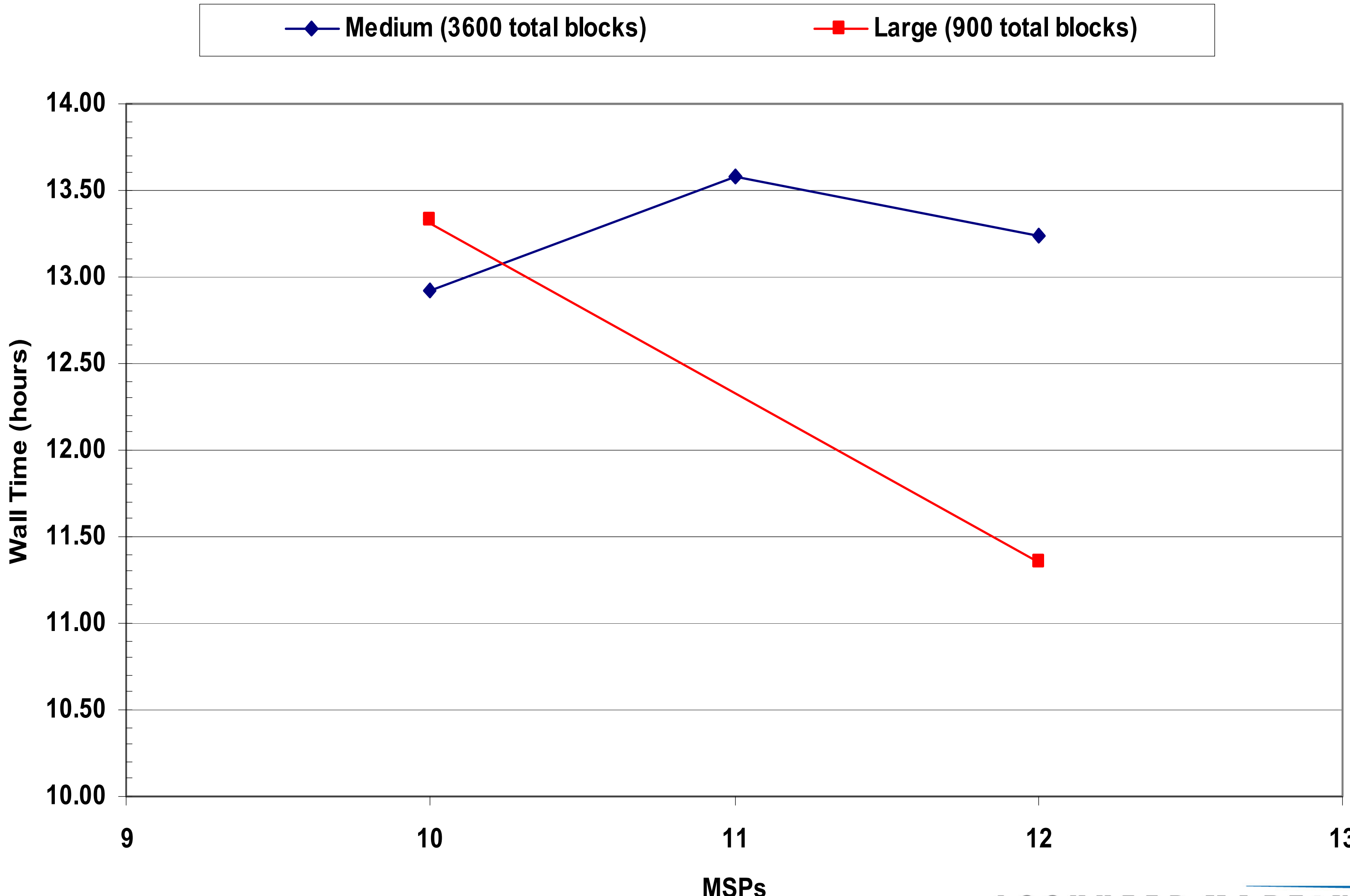


CUG 2004

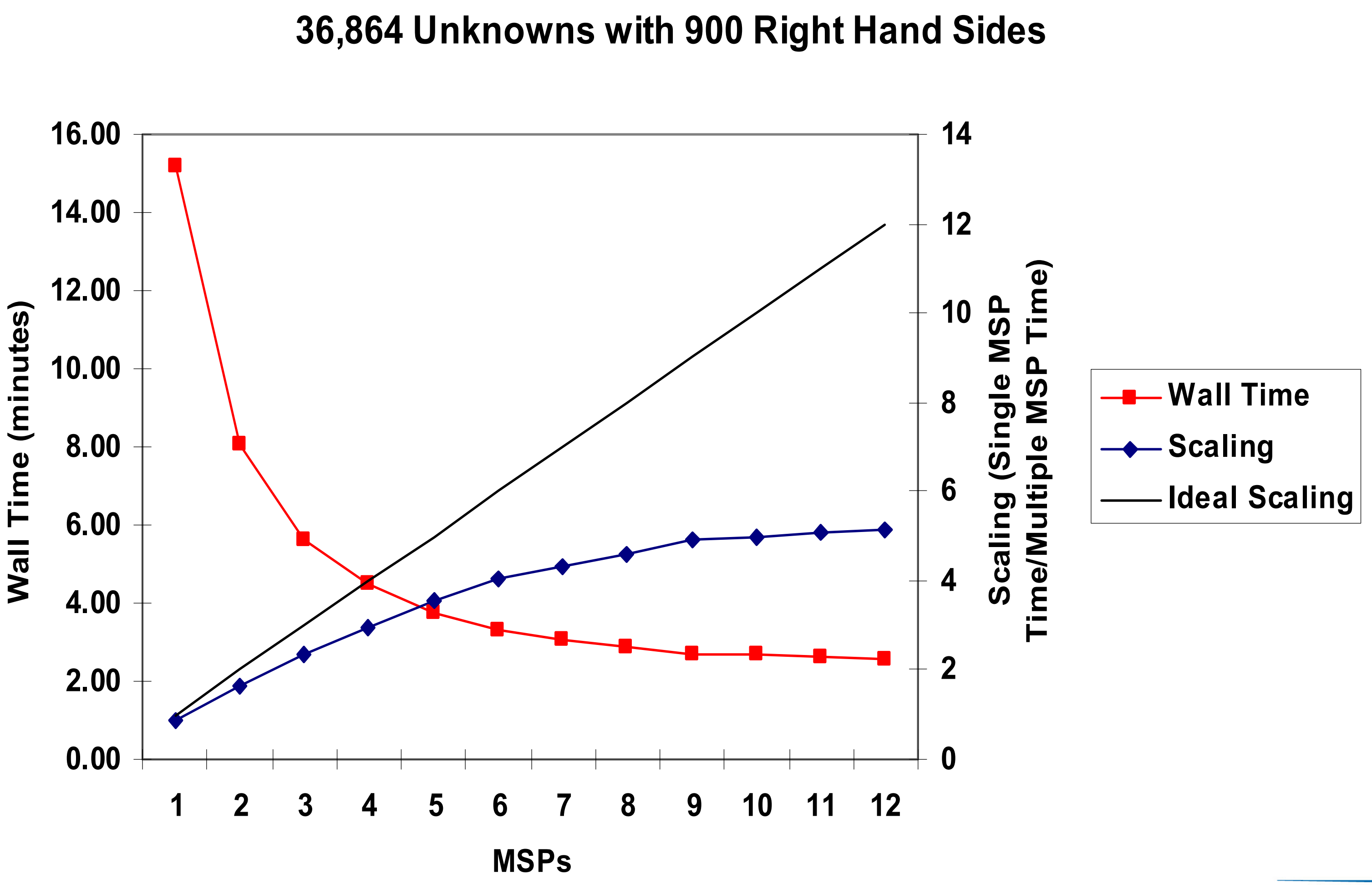
LOCKHEED MARTIN

CRAY

Changing the Block Size – 122,880 Unknowns



Block Solve Benchmarks



Problems

- Software
  - I/O too small – fixed
  - sync_file – fixed?
  - AIO off node – fixed
  - MPT problem - fixed
  - 16 MB direct I/O limit – not fixed
  - 50% of peak bandwidth – not fixed
  - BPT errors – not fixed
- Hardware
  - Like with any new architecture, AHPCRC's X1 was down quite a bit early on (last spring/summer) some due to upgrades

Acknowledgments

- Thanks to the AHPCRC for allowing us to use their Cray X1
- Thanks to many people at Cray for all the help and support

CUG 2004

Back Up Slides

LOCKHEED MARTIN

CRAY

Why Amount of I/O Decreases as Block Size Increases

- Number of I/O requests scales as $N^{3/2}/B$
- Total I/O = number of requests*block size
- 16 total blocks – 1 MB each
  - Number of requests = $16^{1.5}$ = 64
  - **Total I/O = 64*1 MB = 64 MB**
- 4 total blocks – 4 MB each
  - Number of requests = $4^{1.5}$ = 8
  - **Total I/O = 8*4 MB = 32 MB**